1 TD 4 : Résolution d'un Sudoku

(correction page ??)

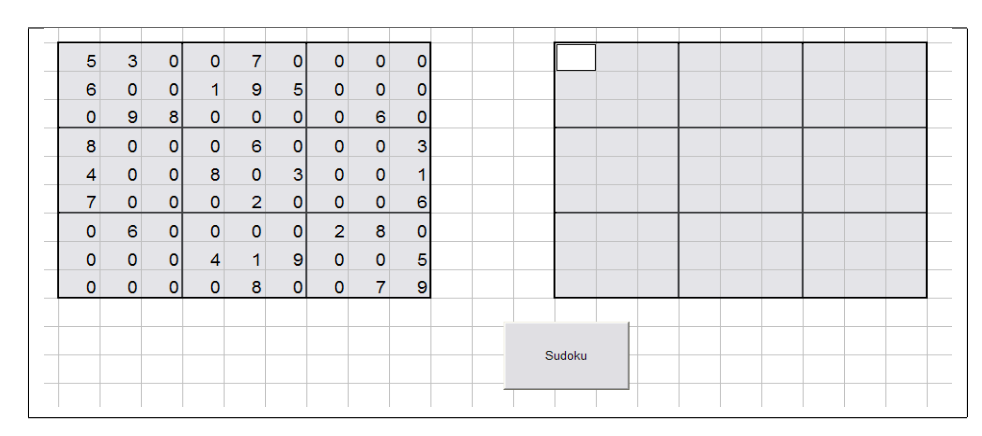
Réjoui(e) ou agacé(e) à l'idée de résoudre un Sudoku ? Et d'utiliser un ordinateur pour le résoudre à votre place ? C'est pourtant ce qui va vous arriver durant les deux heures qui suivent. La figure 1.1 montre ce qu'est un Sudoku. Ce carré incomplet qu'il s'agit de remplir obéit à certaines règles :

1. Chaque case contient un chiffre entre 1 et 9.
2. Dans chaque ligne, il ne peut y avoir deux chiffres égaux.
3. Dans chaque petit carré, il ne peut y deux chiffres égaux.

5	3	0	0	7	0	0	0	0
6	0	0	1	9	5	0	0	0
0	9	8	0	0	0	0	6	0
8	0	0	0	6	0	0	0	3
4	0	0	8	0	3	0	0	1
7	0	0	0	2	0	0	0	6
0	6	0	0	0	0	2	8	0
0	0	0	4	1	9	0	0	5
0	0	0	0	8	0	0	7	9

FIG. 1.1 – Exemple de Sudoku.

A l'aide de ces informations, on va essayer de construire un programme sous Excel qui résoud les Sudoku. La figure 1.2 donne un exemple du résultat qu'on cherche à obtenir. On sélectionne la plage contenant le Sudoku, on sélectionne la plage devant recevoir le résultat, on clique sur le bouton et le tour est joué ou presque.



5	3	0	0	7	0	0	0	0
6	0	0	1	9	5	0	0	0
0	9	8	0	0	0	0	6	0
8	0	0	0	6	0	0	0	3
4	0	0	8	0	3	0	0	1
7	0	0	0	2	0	0	0	6
0	6	0	0	0	0	2	8	0
0	0	0	4	1	9	0	0	5
0	0	0	0	8	0	0	7	9

FIG. 1.2 – Mode d'emploi : le Sudoku est défini par un chiffre en 1 et 9 si celui-ci est connu, 0 s'il est inconnu. On sélectionne à la souris la première plage, on appuie ensuite sur la touche *Ctrl* et on sélection encore à la souris la second plage. Il ne reste plus qu'à cliquer sur le bouton toujours à la souris.

Comment écrire ce programme ? On pourrait choisir une case vide, y poser un chiffre au hasard mais différent de ceux déjà présents sur la même ligne et dans le petit carré auquel appartient la case vide.

Une fois cette case remplie, on obtient un nouveau Sudoku qu'on cherche à nouveau à résoudre. Cette constatation nous dirige vers une fonction récursive qu'on appellera `resolution`. On peut résumer cette fonction en trois étapes :

1. on choisit une case vide
2. on place dans cette case un chiffre qui vérifie les règles d'un Sudoku
3. si toutes les cases sont pleines, c'est fini, sinon on rappelle à nouveau la fonction `resolution` sur ce nouveau Sudoku qui contient une case vide de moins.

Ce découpage suggère trois tâches bien définies. Choisir une case vide peut-être fait aléatoirement dans un premier temps. Nous verrons plus tard comment faire mieux. Pour l'instant, nous allons nous contenter d'un programme simple, qui marche même s'il est lent. La seconde étape maintenant : placer un chiffre dans une case vide, à choisir entre 1 et 9 ou mieux encore dans l'ensemble des chiffres non déjà présents sur la même ligne ou dans le petit carré. Nous allons donc écrire une fonction qui retourne l'ensemble des chiffres possibles pour une case vide. Cette fonction se présente sous cette forme :

```
Function nombre_possible_pour_case(ByRef su As Variant, _
ByVal i As Long, ByVal j As Long) As Variant
    ' ....
End Function
```

Cette fonction prend comme paramètres :

1. `su` : un tableau à deux dimensions, `su(k, l)` désigne la case ligne `k`, colonne `l`
2. `i`, `j` : deux coordonnées désignant la case pour laquelle on veut déterminer l'ensemble des chiffres possibles

Le type `Variant` désigne un type inconnu, en particulier un tableau à deux dimensions. Il existe un nom précis pour tous les types de variables simples `Long`, `String`, `Double`. Les autres comme les tableaux sont souvent désignés par le type `Variant`.

1) La première chose à faire est de vérifier que la case `i, j` est vide. La fonction `nombre_possible_pour_case` doit donc commencer par ces lignes :

```
Function nombre_possible_pour_case(ByRef su As Variant, _
ByVal i As Long, ByVal j As Long) As Variant
    Dim res() As Long     ' on crée le résultat sans connaître son contenu
    If ...........
        ReDim res(0)                    ' on crée un tableau vide
        nombre_possible_pour_case = res ' c'est le résultat de la fonction
        Exit Function                   ' on quitte la fonction
    End If
End Function
```

A vous de trouver la condition à écrire sur la troisième ligne. Si cette case n'est pas vide, il n'y a pas de chiffres possibles pour cette case et le résultat de la fonction est un tableau vide. C'est ce que font les trois lignes suivantes.

2) Si la case est vide, on doit déterminer l'ensemble des chiffres possibles pour cette case. Ou plutôt, ne serait-il pas plus facile de recenser l'ensemble des chiffres qui ne sont pas possibles, c'est-à-dire l'ensemble des chiffres déjà placés soit sur la même ligne, soit dans le même petit carré ? Il suffit pour cela de parcourir la ligne et le petit carré qui contiennent la case `i, j` et de noter tous les chiffres qu'on y rencontre. On crée donc un tableau de neuf cases qui contient `0` si le chiffre est possible et `1` si le chiffre n'est pas possible.

```
Dim paspossible(9) As Long
Dim k As Long
For k = 1 To 9
    paspossible(k) = 0  ' au départ, tous les chiffres sont possibles
Next k
```

Il faut maintenant éliminer tous les chiffres déjà présents sur la ligne `i`. On rappelle que les cases `su(k, l)` sont soit égales à `0` si elles sont vides, soit leur valeur est comprise entre `1` et `9`. C'est à vous d'écrire et de modifier le tableau `paspossible`. Quelques indices : il y a une boucle et un test.

3) On doit maintenant éliminer les chiffres présents dans le petit carré. On calcule pour cela deux entiers `ii`, `jj` grâce aux trois lignes qui suivent :

```
Dim ii, jj As Long
ii = i - ((i - 1) Mod 3)
jj = j - ((j - 1) Mod 3)
```

Que vaux `ii` lorsque `i` varie entre `1` et `9` ? Et `jj` ? Ne pourrait-on pas en déduire quelques lignes permettant d'éliminer les chiffres déjà présents dans le petit carré qui contient la case `i, j` ?

4) Maintenant qu'on a éliminé tous les chiffres pas possibles, on peut construire l'ensemble des chiffres possibles. On va d'abord les compter :

```
Dim n As Long
n = 0
For k = 1 To 9
    If ............... Then n = n + 1
Next k
```

Le résultat attendu est stocké dans la variable `n`. Il suffit juste d'écrire la bonne condition.

5) Il reste plus qu'à construire le résultat `res` de la fonction en complétant le bout de programme suivant :

```
ReDim res(n)
n = 0
For k = 1 To 9
    If ................  Then
        ...........
        ...........
    End If
Next k

' fin
nombre_possible_pour_case = res
```

Petite remarque : si on n'avait pas d'abord compté le nombre de chiffres pas possibles, on n'aurait pas pu créer le bon nombre de cases dans le tableau `res` (première ligne de l'extrait qui précède ce paragraphe).

6) La fonction `nombre_possible_pour_case` retourne donc l'ensemble des chiffres possibles pour une case vide d'un Sudoku. On va d'abord considérer que la case vide qu'on choisit de remplir est la première case vide trouvée. Il ne reste plus qu'à écrire la fonction récursive `resolution` : c'est la fonction qui résoudra le Sudoku. Elle prend comme entrée un tableau Sudoku et retourne un tableau Sudoku pour la solution. On choisit comme convention que ce tableau sera vide si la solution n'existe pas.

```
Function resolution(ByRef su As Variant) As Variant
    ' .....
End Function
```

Il y a trois étapes :

1. On trouve la première case vide.
2. On calcule l'ensemble des chiffres possibles.
3. Pour tous les chiffres possibles, on rappelle la fonction `resolution` avec une case vide en moins.

Deux questions : quand peut-on savoir que le Sudoku est résolu ? Est-il possible d'avoir un ensemble de chiffres possibles vides ? A quoi cela correspond-il et que fait-on dans ce cas-là ?

7) Ecrire le code qui permet de trouver la première case vide et écrivez ce que la fonction doit faire si cette case n'existe pas. N'hésitez pas à lire les deux questions suivantes.

8) Appelez la fonction `nombre_possible_pour_case` pour connaître l'ensemble des chiffres possibles. Que fait-on si l'ensemble est vide ? La fonction `UBound` permet de savoir si un tableau est vide ou non, regardez l'aide pour comprendre comment s'en servir.

9) La fonction `resolution` doit maintenant ressembler à quelque chose comme :

```
Function resolution(ByRef su As Variant) As Variant
    ' étape 1
    Dim i,j,vi,vj As Long
    vi = -1
    For i = 1 To 9
        For j = 1 To 9
            If su (i,j) = 0 Then
                ..................
            End If
        Next j
     Next i

     If vi = ..... Then
         resolution = ..........
         Exit Function
     End If

     ' étape 2
     Dim ens As Variant
     ens = nombre_possible_pour_case (.......)
     If UBound (ens) ...... Then
         Dim res(0) As Long
         resolution = res
         Exit Function
     End If

     ' étape 3
     Dim k As Long
     Dim copie,solution As Variant
     copie = su
     For k = 1 To .....
```

```
        copie ( ...... ) = ens (k)
        solution = resolution (copie)
        If ..... Then
            .....
            Exit Function
        End If
    Next k

    ' au fait, a-t-on réussi ou non si on arrive ici,
    ' lors de l'exécution du programme ?
    resolution = .....
End Function
```

Surtout, enregistrer votre programme avant chaque exécution : si votre programme est entré dans une boucle infini, il faudra fermer Excel de manière autoritaire (en utilisant le gestionnaire des tâches). C'est pourquoi, on propose d'ajouter une variable qui va compter le nombre d'itérations ou d'appels à la fonction `resolution`.

```
Dim nbiter As Long
Function resolution(ByRef su As Variant) As Variant
    nbiter = nbiter + 1
End Function
```

10) Est-il possible de choisir une meilleure case que la première vide ? Et si parmi toutes les cases vides, il y en avait une pour laquelle l'ensemble des chiffres possibles est vide ou ne contient qu'un seul élément ? Comment modifier la fonction `resolution` pour tenir compte de cette information et accélérer la résolution ?

11) Il ne reste plus qu'à extraire le Sudoky depuis une feuille de calcul pour l'envoyer à la fonction `resolution`, ce qui n'est pas forcément une mince affaire. Comme d'habitude, on écrit une macro qu'on pourra relier plus tard à un bouton comme lors des séances précédentes. On vérifie d'abord que l'utilisateur a bien sélectionné 81*2 cases. En VBA, l'ensemble des cases sélectionnées est `Selection`.

```
Sub macro_sudoku()
    Dim i As Long
    i = 1
    For Each ch In Selection
        i = i + 1  ' pointeur d'arrêt ici
    Next ch
    If i <> 81 * 2 Then
        MsgBox "Vous n'avez pas sélectionné 81 * 2 cases"
        Exit Sub
    End If
End Sub
```

Après avoir recopié ce petit bout de programme, placez un pointeur d'arrêt sur la cinquième ligne et regardez le contenu de `ch`. Déduisez-en le moyen d'accéder au contenu d'une case et la manière dont VBA parcourt l'ensemble des cases sélectionnées (en ligne ou en colonne ?).

12) Il faut maintenant récupérer le contenu des 81 premières cases de la sélection pour construire le tableau du Sudoku. Il y a plusieurs manières de le faire, voici l'esquisse d'une.

```
Dim sudoku(9, 9) as Long
i = 1
j = 1
For Each ch In Selection
    sudoku(i, j) = ........
    If j = 9 Then
        ......
    Else
        j = j + 1
    End If
Next ch
```

13) On lance la résolutation.

```
Dim r As Variant
nbiter = 0
r = resolution(sudoku)
```

14) Il n'y a plus qu'à remplir les 81 cases suivantes de la sélection avec la solution.

15) Deux choix s'offrent à vous dorénavant. Le premier consiste à modifier le programme pour obtenir tous les Sudoku qui vérifient la configuration de départ. Le second choix dépend de votre amour inconditionnel des Sudoku, un amour qui vous enjoint de saboter délibérément votre programme pour que jamais il n'ose encore une fois vous répondre en dix fois moins de temps qu'il ne vous en faudrait pour terminer votre Sudoku. Cela dit, si vous comptiez vous rabattre sur le démineur, il est sans doute heureux qu'il n'y ait pas de séance de VBA supplémentaire.